LÓGICA DE PROGRAMAÇÃO

Para Iniciantes

Do Pensamento ao Algoritmo — Aprenda a Programar do Zero

Atividade de Extensão — UNIP
Curso: Análise e Desenvolvimento de Sistemas
Área Temática: Tecnologia e Produção
2024

SUMÁRIO

Apresentação 3
Capítulo 1 — O que é Lógica de Programação? 4
1.1 Algoritmos no cotidiano 4
1.2 Por que aprender lógica antes de uma linguagem? 4
Capítulo 2 — Variáveis e Tipos de Dados 5
2.1 O que é uma variável? 5
2.2 Tipos de dados primitivos 5
2.3 Regras para nomear variáveis 6
Capítulo 3 — Operadores 7
3.1 Operadores aritméticos 7
3.2 Operadores relacionais 7
3.3 Operadores lógicos 8
Capítulo 4 — Estruturas de Decisão 9
4.1 SE / ENTÃO / SENÃO (if / else) 9
4.2 SE aninhado (if / else if) 10
4.3 ESCOLHA / CASO (switch) 10
Capítulo 5 — Estruturas de Repetição 11
5.1 ENQUANTO (while) 11
5.2 FAÇA ENQUANTO (do while) 12
5.3 PARA (for) 12
5.4 Break e Continue 13
Capítulo 6 — Vetores e Matrizes 14
6.1 O que é um vetor (array)? 14
6.2 Operações com vetores 14
6.3 Matrizes (array bidimensional) 15
Capítulo 7 — Funções e Procedimentos 16
7.1 O que é uma função? 16
7.2 Parâmetros e retorno 16
7.3 Escopo de variáveis 17
Capítulo 8 — Algoritmos Clássicos 18
8.1 Algoritmo de busca linear 18
8.2 Algoritmo de ordenação (Bubble Sort) 18
8.3 Cálculo de fatorial (recursão) 19
Capítulo 9 — Pseudocódigo e Fluxogramas 20
9.1 Pseudocódigo 20
9.2 Fluxograma — símbolos e exemplos 21
Resumo Final e Próximos Passos 22

Apresentação

Este e-book foi desenvolvido como Atividade de Extensão do curso Superior Tecnológico em Análise e Desenvolvimento de Sistemas da Universidade Paulista – UNIP, com o objetivo de compartilhar conhecimento sobre Lógica de Programação de forma acessível e gratuita para a comunidade.

A Lógica de Programação é a base de toda a computação. Antes de escrever código em qualquer linguagem — Python, JavaScript, Java ou outra — é fundamental entender como estruturar o raciocínio para resolver problemas de forma sistemática. Sem essa base, aprender uma linguagem de programação é como tentar construir uma casa sem alicerce.

Este material foi escrito para quem está começando do zero. Não é necessário nenhum conhecimento prévio — apenas curiosidade e vontade de aprender. Os exemplos são apresentados em pseudocódigo (linguagem próxima ao português) e em JavaScript, tornando a leitura acessível a qualquer pessoa.

O que você vai aprender

Variáveis, tipos de dados, operadores, estruturas de decisão (if/else/switch), estruturas de repetição (while/for/do-while), vetores, matrizes, funções, algoritmos clássicos, pseudocódigo e fluxogramas — tudo com exemplos práticos e exercícios resolvidos.

Capítulo 1 — O que é Lógica de Programação?

1.1 Algoritmos no cotidiano

Lógica de Programação é a capacidade de organizar um raciocínio em passos sequenciais, claros e sem ambiguidade, para resolver um problema. Esse conjunto de passos é chamado de algoritmo.

Você já cria algoritmos no dia a dia sem perceber. Veja o algoritmo para fritar um ovo:

1. Ligar o fogão
2. Colocar a frigideira no fogo
3. Adicionar óleo
4. Esperar o óleo aquecer
5. Quebrar o ovo na frigideira
6. Esperar fritar
7. Tirar e servir

Perceba: os passos têm ordem, são específicos e levam a um resultado. É exatamente assim que funciona um programa de computador.

1.2 Por que aprender lógica antes de uma linguagem?

Muitos iniciantes cometem o erro de tentar aprender Python ou JavaScript sem antes dominar a lógica. O resultado é frustração — o código não funciona e a pessoa não sabe por quê.

A linguagem de programação é apenas o idioma em que você escreve o algoritmo. Se você domina a lógica, aprender qualquer linguagem é questão de conhecer a sintaxe. Por isso, neste e-book usamos pseudocódigo (linguagem em português próxima à programação) e exemplos simples em JavaScript.

> **Analogia**
> Lógica de programação é como aprender a pensar musicalmente antes de tocar um instrumento. Você pode trocar o instrumento (linguagem), mas a teoria (lógica) permanece a mesma.

Capítulo 2 — Variáveis e Tipos de Dados

2.1 O que é uma variável?

Uma variável é um espaço na memória do computador onde guardamos um valor que pode mudar ao longo do programa. Pense nela como uma caixa com um nome, onde você coloca e retira informações.

Exemplo em pseudocódigo:

```
// Declarando variáveis
inteiro idade = 25
texto nome = "Maria"
real salario = 3500.50
logico ativo = verdadeiro
```

Exemplo em JavaScript:

```
let idade = 25;
let nome = "Maria";
let salario = 3500.50;
let ativo = true;

console.log(nome);   // Saída: Maria
console.log(idade);  // Saída: 25
```

2.2 Tipos de dados primitivos

Tipo	Pseudocódigo / JS	Exemplo
Inteiro	`inteiro / Number`	idade = 25
Real	`real / Number`	salario = 3500.50
Texto	`texto / String`	nome = "Ana"
Lógico	`logico / Boolean`	ativo = true
Nulo	`nulo / null`	valor = null
Indefinido	`— / undefined`	let x;

> **Number no JavaScript**

> No JavaScript, inteiros e reais são todos do tipo Number. A diferença entre 5 e 5.0 é apenas visual — o tipo é o mesmo.

2.3 Regras para nomear variáveis

- Comece com letra ou underscore (_): nomeCliente, _valor
- Não use espaços: use camelCase (nomeCompleto) ou snake_case (nome_completo)
- Não use palavras reservadas da linguagem: if, for, while, let, etc.
- Use nomes descritivos: idadeAluno é melhor que x
- Diferencie maiúsculas de minúsculas: Idade e idade são variáveis diferentes

```
// Nomes corretos
let nomeCliente = "João";
let totalVendas = 0;
let _auxiliar = true;

// Nomes incorretos
let 1nome = "erro";         // começa com número
let nome completo = "erro"; // tem espaço
let let = "erro";           // palavra reservada
```

Constantes — valores que não mudam

Use const para valores fixos. Por convenção, escreva em MAIÚSCULAS:

```
const PI = 3.14159;
const TAXA_JUROS = 0.05;
const NOME_EMPRESA = "UNIP";
```

Capítulo 3 — Operadores

3.1 Operadores Aritméticos

Usados para realizar cálculos matemáticos:

Operador	Símbolo	Exemplo e Resultado
Adição	+	5 + 3 → 8
Subtração	-	10 - 4 → 6
Multiplicação	*	3 * 4 → 12
Divisão	/	10 / 2 → 5
Módulo (resto)	%	10 % 3 → 1
Potenciação	**	2 ** 3 → 8

```
let a = 10, b = 3;

console.log(a + b);   // 13
console.log(a - b);   // 7
console.log(a * b);   // 30
console.log(a / b);   // 3.333...
console.log(a % b);   // 1  (resto da divisão)
console.log(a ** b);  // 1000  (10³)
```

Operadores de atribuição composta

```
let x = 10;
x += 5;   // x = x + 5  →  x = 15
x -= 3;   // x = x - 3  →  x = 12
x *= 2;   // x = x * 2  →  x = 24
x /= 4;   // x = x / 4  →  x = 6
x++;      // x = x + 1  →  x = 7  (incremento)
x--;      // x = x - 1  →  x = 6  (decremento)
```

3.2 Operadores Relacionais

Comparam valores e retornam verdadeiro (true) ou falso (false):

Operador	Símbolo	Exemplo (a=5, b=3)
Igual	==	a == 5 → true

Estritamente igual	`===`	5 === '5' → false
Diferente	`!=`	a != b → true
Maior que	`>`	a > b → true
Menor que	`<`	a < b → false
Maior ou igual	`>=`	a >= 5 → true
Menor ou igual	`<=`	a <= 3 → false

> **== vs === no JavaScript**
> Use sempre === (igualdade estrita). O == faz conversão de tipo e pode gerar resultados inesperados: 5 == '5' é verdadeiro, mas 5 === '5' é falso. Prefira === para evitar bugs.

3.3 Operadores Lógicos

Combinam expressões booleanas:

Operador	Símbolo	Regra
E (AND)	`&&`	Verdadeiro se AMBAS forem verdadeiras
OU (OR)	`||`	Verdadeiro se PELO MENOS UMA for verdadeira
NÃO (NOT)	`!`	Inverte o valor: !true = false

```
let idade = 20;
let temCarteira = true;

// AND — ambas devem ser verdadeiras
console.log(idade >= 18 && temCarteira);  // true

// OR — pelo menos uma verdadeira
console.log(idade < 18 || temCarteira);   // true

// NOT — inverte
console.log(!temCarteira);                // false
```

Capítulo 4 — Estruturas de Decisão

Estruturas de decisão permitem que o programa escolha qual caminho seguir com base em uma condição. São o cérebro do algoritmo.

4.1 SE / ENTÃO / SENÃO — if / else

A estrutura mais básica de decisão: se a condição for verdadeira, executa um bloco; senão, executa outro.

```
// Pseudocódigo
SE (condicao) ENTAO
   // bloco executado se verdadeiro
SENAO
   // bloco executado se falso
FIM SE
```

```
// JavaScript
let nota = 7;

if (nota >= 6) {
   console.log("Aprovado!);
} else {
   console.log("Reprovado);
}
// Saída: Aprovado!
```

4.2 SE Aninhado — if / else if

Quando há mais de duas possibilidades, encadeamos vários SE:

```
let nota = 7.5;

if (nota >= 9) {
   console.log("Conceito A — Excelente");
} else if (nota >= 7) {
   console.log("Conceito B — Bom");
} else if (nota >= 5) {
   console.log("Conceito C — Regular");
} else {
   console.log("Reprovado");
}
```

```
// Saída: Conceito B — Bom
```

4.3 ESCOLHA / CASO — switch

O switch é ideal quando você compara uma variável com vários valores fixos possíveis:

```
let diaSemana = 3;

switch (diaSemana) {
   case 1: console.log("Segunda-feira"); break;
   case 2: console.log("Terça-feira");   break;
   case 3: console.log("Quarta-feira");  break;
   case 4: console.log("Quinta-feira");  break;
   case 5: console.log("Sexta-feira");   break;
   case 6: console.log("Sábado");        break;
   case 7: console.log("Domingo");       break;
   default: console.log("Dia inválido");
}
// Saída: Quarta-feira
```

Não esqueça o break!
Sem o break, o switch continua executando os casos seguintes mesmo que já tenha encontrado o correto. Isso é chamado de 'fall-through' e gera resultados inesperados.

Capítulo 5 — Estruturas de Repetição

Estruturas de repetição (também chamadas de laços ou loops) permitem executar um bloco de código várias vezes, evitando repetição manual. São essenciais em programação.

5.1 ENQUANTO — while

Executa o bloco enquanto a condição for verdadeira. A condição é verificada ANTES de cada execução.

```
// Pseudocódigo
ENQUANTO (condicao) FACA
   // bloco repetido
FIM ENQUANTO
```

```
// Contar de 1 a 5
let contador = 1;

while (contador <= 5) {
   console.log("Número: " + contador);
   contador++;   // IMPORTANTE: atualizar ou loop infinito!
}
// Saída: Número: 1, Número: 2, ..., Número: 5
```

> **Cuidado com loop infinito!**
> Se a condição do while nunca se tornar falsa, o programa trava. Certifique-se sempre de que a variável de controle é atualizada dentro do loop.

5.2 FAÇA ENQUANTO — do while

Semelhante ao while, mas a condição é verificada DEPOIS de cada execução. Isso garante que o bloco execute pelo menos uma vez.

```
let numero;

do {
   numero = parseInt(prompt("Digite um número positivo:"));
} while (numero <= 0);

console.log("Número válido: " + numero);
```

> **Quando usar do while?**
> Use do while quando o bloco precisa executar pelo menos uma vez, independente da condição. O exemplo acima é clássico: validar entrada do usuário — você precisa pedir o número antes de validar.

5.3 PARA — for

O for é ideal quando você sabe exatamente quantas vezes o loop deve executar. Ele concentra inicialização, condição e incremento em uma linha:

```
// Estrutura: for (início; condição; incremento)
for (let i = 1; i <= 5; i++) {
   console.log("Iteração: " + i);
}
// Saída: Iteração: 1, 2, 3, 4, 5
```

```
// Somando números de 1 a 100
let soma = 0;

for (let i = 1; i <= 100; i++) {
   soma += i;
}

console.log("Soma: " + soma);  // Soma: 5050
```

5.4 Break e Continue

Break — interrompe o loop

```
// Para ao encontrar o número 4
for (let i = 1; i <= 10; i++) {
   if (i === 4) break;
   console.log(i);
}
// Saída: 1, 2, 3
```

Continue — pula para a próxima iteração

```
// Pula números pares — imprime só ímpares
for (let i = 1; i <= 10; i++) {
```

```
    if (i % 2 === 0) continue;
    console.log(i);
}
// Saída: 1, 3, 5, 7, 9
```

Estrutura	Quando usar
`while`	Número de repetições desconhecido / depende de condição
`do while`	Bloco deve executar pelo menos 1 vez
`for`	Número de repetições conhecido / iterar sequências

Capítulo 6 — Vetores e Matrizes

6.1 O que é um vetor (array)?

Um vetor (ou array) é uma variável que armazena vários valores do mesmo tipo em sequência, identificados por um índice numérico. Em vez de criar 10 variáveis para guardar 10 notas, usamos um vetor.

> **Índices começam em 0**
> Em programação, o primeiro elemento de um vetor está na posição 0, não na posição 1. Um vetor com 5 elementos tem índices de 0 a 4.

```
// Declarando um vetor de notas
let notas = [8.5, 7.0, 9.2, 6.8, 7.5];

console.log(notas[0]);  // 8.5  (primeiro elemento)
console.log(notas[2]);  // 9.2  (terceiro elemento)
console.log(notas[4]);  // 7.5  (último elemento)
console.log(notas.length);  // 5  (tamanho do vetor)
```

6.2 Operações com vetores

Percorrendo um vetor com for

```
let notas = [8.5, 7.0, 9.2, 6.8, 7.5];
let soma = 0;

for (let i = 0; i < notas.length; i++) {
   soma += notas[i];
}

let media = soma / notas.length;
console.log("Média: " + media.toFixed(2));  // Média: 7.80
```

Métodos úteis de array em JavaScript

Método	O que faz
`push(valor)`	Adiciona elemento no final
`pop()`	Remove e retorna o último elemento
`unshift(valor)`	Adiciona elemento no início

`shift()`	Remove e retorna o primeiro elemento
`indexOf(valor)`	Retorna o índice do elemento (-1 se não encontrar)
`includes(valor)`	Retorna true se o elemento existir
`sort()`	Ordena o array (atenção: modifica o original)
`length`	Retorna o número de elementos

6.3 Matrizes (array bidimensional)

Uma matriz é um vetor de vetores — uma tabela com linhas e colunas. Útil para representar dados tabulares, grades, tabuleiros de jogos, etc.

```
// Matriz 3x3 — notas de 3 alunos em 3 provas
let notas = [
   [8.0, 7.5, 9.0],   // Aluno 1
   [6.0, 8.5, 7.0],   // Aluno 2
   [9.5, 9.0, 8.0],   // Aluno 3
];

console.log(notas[0][1]);  // 7.5 (Aluno 1, Prova 2)
console.log(notas[2][0]);  // 9.5 (Aluno 3, Prova 1)

// Percorrendo com for aninhado
for (let i = 0; i < notas.length; i++) {
   for (let j = 0; j < notas[i].length; j++) {
      process.stdout.write(notas[i][j] + "  ");
   }
   console.log("");
}
```

Capítulo 7 — Funções e Procedimentos

7.1 O que é uma função?

Uma função é um bloco de código com nome próprio que realiza uma tarefa específica e pode ser reutilizado quantas vezes for necessário. Funções tornam o código organizado, legível e sem repetição.

> **Princípio da responsabilidade única**
> Uma boa função deve fazer UMA coisa bem feita. Se a função está muito longa ou faz muitas coisas diferentes, considere dividi-la em funções menores.

```
// Declarando uma função simples
function saudar() {
   console.log("Olá! Bem-vindo ao mundo da programação!");
}

// Chamando a função
saudar();  // Saída: Olá! Bem-vindo ao mundo da programação!
saudar();  // Pode chamar quantas vezes quiser
```

7.2 Parâmetros e retorno

Funções podem receber dados de entrada (parâmetros) e devolver um resultado (retorno):

```
// Função com parâmetros e retorno
function somar(a, b) {
   return a + b;
}

let resultado = somar(5, 3);
console.log(resultado);  // 8
console.log(somar(10, 20));  // 30
```

```
// Função mais completa — calcular média
function calcularMedia(notas) {
   let soma = 0;
   for (let i = 0; i < notas.length; i++) {
      soma += notas[i];
   }
```

```
    return soma / notas.length;
}

let minhasNotas = [8.0, 7.5, 9.0, 6.5];
let media = calcularMedia(minhasNotas);
console.log("Média: " + media.toFixed(2));  // Média: 7.75
```

Arrow Functions — sintaxe moderna

```
// Forma tradicional
function quadrado(x) {
    return x * x;
}

// Arrow function equivalente
const quadrado = (x) => x * x;

console.log(quadrado(4));   // 16
console.log(quadrado(10));  // 100
```

7.3 Escopo de variáveis

Escopo define onde uma variável pode ser acessada no código:

```
let global = "Sou global";   // acessível em qualquer lugar

function minhaFuncao() {
    let local = "Sou local";  // acessível só dentro da função
    console.log(global);       // □ funciona
    console.log(local);        // □ funciona
}

minhaFuncao();
console.log(global);   // □ funciona
console.log(local);    // □ erro! local não existe aqui
```

Capítulo 8 — Algoritmos Clássicos

8.1 Algoritmo de Busca Linear

A busca linear percorre cada elemento de um array até encontrar o valor procurado. É simples, mas eficiente para listas pequenas.

```
function buscaLinear(vetor, alvo) {
   for (let i = 0; i < vetor.length; i++) {
      if (vetor[i] === alvo) {
         return i;  // retorna o índice onde encontrou
      }
   }
   return -1;  // retorna -1 se não encontrou
}

let numeros = [15, 3, 42, 8, 27, 11];
console.log(buscaLinear(numeros, 27));   // 4 (índice)
console.log(buscaLinear(numeros, 99));   // -1 (não encontrado)
```

8.2 Algoritmo de Ordenação — Bubble Sort

O Bubble Sort compara pares de elementos adjacentes e os troca se estiverem na ordem errada, repetindo até o array estar ordenado. É o algoritmo de ordenação mais didático para iniciantes.

```
function bubbleSort(vetor) {
   let n = vetor.length;
   for (let i = 0; i < n - 1; i++) {
      for (let j = 0; j < n - i - 1; j++) {
         if (vetor[j] > vetor[j + 1]) {
            // Troca os elementos de posição
            let temp = vetor[j];
            vetor[j] = vetor[j + 1];
            vetor[j + 1] = temp;
         }
      }
   }
   return vetor;
}

let numeros = [64, 34, 25, 12, 22, 11, 90];
console.log(bubbleSort(numeros));
// Saída: [11, 12, 22, 25, 34, 64, 90]
```

Como o Bubble Sort funciona?
Imagine bolhas subindo em água: os valores maiores 'borbulham' para o final do array a cada passagem. Após N-1 passagens, o array estará ordenado do menor para o maior.

8.3 Fatorial — Introdução à Recursão

Recursão é quando uma função chama a si mesma. O cálculo de fatorial (n! = n × (n-1) × ... × 1) é o exemplo mais clássico:

```
// Versão iterativa (com loop)
function fatorialIterativo(n) {
   let resultado = 1;
   for (let i = 2; i <= n; i++) {
      resultado *= i;
   }
   return resultado;
}

// Versão recursiva
function fatorialRecursivo(n) {
   if (n <= 1) return 1;         // caso base
   return n * fatorialRecursivo(n - 1);  // chamada recursiva
}

console.log(fatorialIterativo(5));  // 120  (5!=5×4×3×2×1)
console.log(fatorialRecursivo(5));  // 120
```

Capítulo 9 — Pseudocódigo e Fluxogramas

9.1 Pseudocódigo

Pseudocódigo é uma forma de escrever algoritmos em linguagem próxima ao português, sem se preocupar com a sintaxe exata de nenhuma linguagem de programação. É a ferramenta ideal para planejar o algoritmo antes de codificar.

Palavras-chave do Pseudocódigo

Palavra-chave	Equivalente em programação
`INICIO / FIM`	Início e fim do programa
`LEIA`	Entrada de dado (input)
`ESCREVA`	Saída de dado (output/console.log)
`SE / ENTAO / SENAO / FIM SE`	if / then / else / }
`ENQUANTO / FACA / FIM ENQUANTO`	while / { / }
`PARA / ATE / FACA / FIM PARA`	for / to / { / }
`FUNCAO / FIM FUNCAO`	function / }
`RETORNE`	return
`inteiro / real / texto / logico`	number / string / boolean

Exemplo completo em pseudocódigo

Algoritmo para calcular a média de um aluno e informar se foi aprovado:

```
INICIO
   LEIA nome
   LEIA nota1, nota2, nota3

   media <- (nota1 + nota2 + nota3) / 3

   SE (media >= 7.0) ENTAO
      ESCREVA nome, "foi APROVADO com média", media
   SENAO SE (media >= 5.0) ENTAO
      ESCREVA nome, "está em RECUPERAÇÃO com média", media
   SENAO
      ESCREVA nome, "foi REPROVADO com média", media
   FIM SE
```

```
FIM
```

9.2 Fluxograma — Símbolos essenciais

Símbolo	Significado
Oval / Elipse	Início e Fim do algoritmo
Paralelogramo	Entrada (LEIA) ou Saída (ESCREVA) de dados
Retângulo	Processamento / Operação / Atribuição
Losango	Decisão — pergunta com resposta Sim/Não
Seta	Fluxo de direção do algoritmo
Retângulo com listras	Chamada de sub-rotina / função

Fluxograma do algoritmo de média acima:

```
   [INÍCIO]
       |
   [Ler nome, nota1, nota2, nota3]
       |
   [media = (nota1 + nota2 + nota3) / 3]
       |
   <media >= 7?> --SIM--> [Exibe: APROVADO]
       |
      NÃO
       |
   <media >= 5?> --SIM--> [Exibe: RECUPERAÇÃO]
       |
      NÃO
       |
   [Exibe: REPROVADO]
       |
    [FIM]
```

Resumo Final e Próximos Passos

Tabela Resumo — Tudo que você aprendeu

Conceito	O que é
Algoritmo	Sequência de passos para resolver um problema
Variável	Espaço na memória para guardar valores
Tipos de dados	inteiro, real, texto, lógico
Operadores	Aritméticos (+,-,*,/,%), relacionais (==,>,<) e lógicos (&&,||,!)
if / else	Decisão: executa blocos diferentes conforme a condição
switch	Decisão múltipla: compara uma variável com vários valores
while	Repete enquanto a condição for verdadeira (testa antes)
do while	Repete enquanto a condição for verdadeira (testa depois)
for	Repete um número conhecido de vezes
Array (vetor)	Lista de valores acessados por índice
Matriz	Array de arrays — tabela de linhas e colunas
Função	Bloco de código reutilizável com nome
Parâmetro/Retorno	Entrada e saída de dados de uma função
Escopo	Onde uma variável pode ser acessada
Busca Linear	Percorre o array até encontrar o elemento
Bubble Sort	Ordena comparando pares de elementos adjacentes
Recursão	Função que chama a si mesma
Pseudocódigo	Algoritmo escrito em linguagem próxima ao português
Fluxograma	Representação visual de um algoritmo

O que estudar depois?

Com a base deste e-book, você está pronto para avançar. Recomendações de próximos passos:

- Escolha uma linguagem: Python é excelente para iniciantes e tem vasta aplicação em dados e IA
- Aprenda Programação Orientada a Objetos (POO): classes, objetos, herança, encapsulamento
- Estude estruturas de dados: pilhas, filas, listas ligadas, árvores

- Pratique em plataformas gratuitas: Beecrowd (antigo URI Online Judge), HackerRank, LeetCode
- Aprenda Git e controle de versão para trabalhar em equipe
- Explore frameworks web: React (JavaScript) ou Django (Python)

Ferramentas gratuitas para praticar

Ferramenta	Para quê serve
Replit (replit.com)	Programar direto no navegador, sem instalar nada
VS Code (code.visualstudio.com)	Editor de código profissional e gratuito
Node.js (nodejs.org)	Executar JavaScript fora do navegador
Beecrowd (beecrowd.com.br)	Desafios de algoritmos em português
Khan Academy (pt.khanacademy.org)	Cursos gratuitos de programação para iniciantes
CS50 (cs50.harvard.edu)	Curso gratuito de Ciência da Computação de Harvard

Bons estudos e bons algoritmos!

Atividade de Extensão UNIP Análise e Desenvolvimento de Sistemas
Educação de Qualidade | Trabalho Decente e Crescimento Econômico